View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by PDXScholar

Portland State University

PDXScholar

Electrical and Computer Engineering Faculty Publications and Presentations

Electrical and Computer Engineering

6-26-2016

Exploring Proficiency Testing of Programming Skills in Lower-division Computer Science and Electrical Engineering Courses

Karla Steinbrugge Fant
Portland State University, karlaf@pdx.edu

Branimir Pejcinovic
Portland State University, pejcinb@pdx.edu

Phillip Wong
Portland State University, pkwong@pdx.edu

Follow this and additional works at: https://pdxscholar.library.pdx.edu/ece_fac

 Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Citation Details

Karla Fant, Branimir Pejcinovic, and Phillip Wong. Exploring profi- ciency testing of programming skills in lower-division computer science and electrical engineering courses. In 2016 ASEE Annual Conference and Exposition. American Society for Engineering Education, 2016.

This Post-Print is brought to you for free and open access. It has been accepted for inclusion in Electrical and Computer Engineering Faculty Publications and Presentations by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Exploring Proficiency Testing of Programming Skills in Lower-division Computer Science and Electrical Engineering Courses

Motivation

It is generally accepted that all engineering students should be able to perform some programming tasks. For example, ABET calls for electrical engineering (EE) curricula to include "engineering topics (including computing science) necessary to analyze and design complex electrical and electronic devices, software, and systems containing hardware and software components."[1] In most disciplines, programming plays a supporting role as one of the tools that future engineers will need to tackle problem solving and design projects. Because it is considered such a basic tool, programming is typically taught in freshman or sophomore courses. Many engineering students get their first exposure to programming in a class where a programming language is used to assist problem solving. In electrical engineering (EE), this may be followed by another course covering more advanced programming concepts. For example, in our EE program at Portland State University (PSU), we teach MATLAB as part of a first-year "introduction to engineering" and problem solving course. This is then followed by an intermediate level C programming course. Obviously, in computer science (CS) programs there is much greater emphasis on immediate application of programming and development of necessary theoretical concepts. Most engineering programs do not have time in their curriculum to prepare students in their lower-division coursework to a similar level of depth.

Because of the exact and unforgiving nature of programming, many students struggle when asked to perform what appear to be simple programming tasks[2]. This problem has persisted to the present day. At the same time, many students also have difficulty formulating general problem solving strategies, which makes it even harder to use programming as a tool. These and other factors contribute to high attrition rates in freshman engineering courses. It was our observation of students' struggles that led us to consider the question of what the best practices in programming in CS may be and to try to transfer them to our courses.

Brief Introduction to Terminology

In order to provide proper context for our work, we need to provide a brief historical note and explain our terminology. The proficiency or competency based approach to testing of programming skills is not a new concept, and it was discussed and implemented[3] in the 1980s and even earlier. It was also mixed in with the concept of demonstrating "mastery" of a given topic, i.e., programming. For example, Carnegie Mellon University instituted a final exam in which students were supposed to demonstrate that they can accomplish certain tasks in a controlled environment[4,5]. This was meant to provide several benefits, one being that "it was hoped that the Mastery Exam would address a general problem of students successfully passing programming courses at CMU without also learning the rudiments of programming methodology"[4]. Another benefit would be reducing actual or potential cheating on exams. More recent attempts to test student programming skills and its associated problems have been published[6,7]. Various studies utilize different assessment approaches and define their own set of learning outcomes, competencies or skills that are being assessed. Defining what appropriate programming skills are is in itself a difficult problem[8].

More recently, competency-based education has become popular in many different fields[9], and the driving force seems to be "Transitioning away from seat time, in favor of a structure that creates flexibility, allows students to progress as they demonstrate mastery of academic content, regardless of time, place, or pace of learning"[10]. However, despite its long history, the competency-based education area is still very much in flux, and there are many different definitions and labels used. In our case we need to make a distinction between competency and proficiency. In our present view, demonstrating competency is a binary concept – students can either complete a given task or not. Proficiency is a more nuanced assessment, and it may also include assessment of behaviors that students exhibit. In either case, some standard level of performance needs to be defined. In computer science, students are expected to build on the programming foundation throughout their four years of study, so it seems appropriate that a more nuanced approach be taken. In our electrical engineering courses, however, we are more interested in basic understanding and application of programming to problem solving, for which a simpler assessment seems sufficient.

In the sections below we will discuss implementation of proficiency-based testing (PT) across many courses in the computer science program and follow that up with a discussion of competency-based testing (CT) in one freshman electrical engineering course.

Proficiency-based Testing in CS: Background

The core programming competencies expected of Computer Science (CS) undergraduate students are cumulative. The first two years of PSU's CS program builds foundational level material needed by most courses in the curriculum. In regards to programming, students need to meet the programming requirements outlined by a prerequisite course in order to be successful in follow-on courses. For example, one must be proficient using variables in order to progress to loops and functions. And, one must be proficient using pointers and/or references in order to be proficient at programming linked data structures. Ensuring proficiency at each level solidifies the program and enables student success. There is an additional benefit in that students are less frequently taken by surprise in the programming pre-requisites and are less likely to drop.

Understanding computer science concepts in some theoretical sense is not sufficient if students cannot apply what they have learned. Allowing students to progress when they do not have sufficient programming experience creates situations where prerequisite classes have to be taken multiple times or the course difficulty level is reduced so much that the overall curriculum quality suffers. The proficiency-based testing model addresses this issue early on, which minimizes the cost and time impact. A critical part of proficiency testing is determining essential competencies and the standards by which they are evaluated. Once these are determined they communicate a clear and powerful message to students in terms of expectations. We have found that traditional, paper-and-pencil, in-class exams are insufficient on their own. Many students can collect sufficient partial credit or show memorization of facts but still be lacking in actual programming and problem solving skills. Students may think that they have mastered the material but have trouble integrating the concepts together into a finished program. In contrast to a traditional exam, PT is performed in small groups and in a controlled environment. PT exams are observed so that faculty can determine how students approach problem solving and

debugging that cannot be observed otherwise. Similarly, once they are in place, PT examinations can be used to ensure that transfer students have the same abilities as native students.

Based on these arguments we would expect the following benefits to come from using PT:

1. Improved quality of student work
2. Improved courses and curriculum based on direct observational feedback
3. Ensured level of proficiency among native and transfer students
4. Improved formative assessment – students receive immediate and direct feedback
5. Clear communication of expectations to students
6. Triangulation of student performance
7. Established baseline, ensuring that students are prepared for future courses

In addition, we have found out that this approach is scalable to a relatively large number of students, and we have handled up to 450 students per quarter. However, there are organizational issues to be resolved and adequate resources should be provided. A discussion of these benefits and supporting evidence is presented below.

Proficiency-based Testing in CS: History

The Computer Science department at our university officially started administering proficiency-based testing in winter 2012. Students enrolled in the core programming courses at the freshman and sophomore level participate in proficiency examinations twice a term. These require students to demonstrate programming core competencies at the level expected for the given course. Students currently experience proficiency examinations in CS161 Introduction to Programming, CS162 Introduction to Computer Science, CS163 Data Structures, and CS202 Advanced Data structures and OOP. Beginning in summer 2014, transfer students applying to our Computer Science department for entry into the upper division curriculum also began demonstrating competencies with proficiency-based testing. Such proficiency examinations ensure that students from all backgrounds are able to meet the expected core competencies of a computer scientist prior to entry into the upper division.

Determining and Evaluating Student Competencies in CS

Through ABET accreditation, each of the required courses has specific core competencies that students are evaluated upon. Note that during PT, students are scored on the process of problem solving, program design, and coding. PT results are used to assess student programming skills but are not designed to cover all other learning outcomes. Table 1 summarizes the type of scoring used for four of the core competencies. Proficiency levels are scored as:

- E – Exceeds our expectations. The solution came quickly, and it was obvious from the student's approach that they understood the programming platform, editors, debuggers (if applicable), language, syntax, and data structure (if applicable). There is an obvious fluency in how they approach the design and programming problem assigned. The code compiled without syntax errors or warnings.
- P – Proficient meeting our expectations. The process of solving the problem demonstrated the level of proficiency expected for the course. The student understood how to design and implement a problem and was capable using the platform, editors, debuggers (if applicable), language, syntax and data structure (if applicable). They may

have redesigned the solution and re-compiled multiple times. It is clear from our observations that the student does understand how to solve the problem even if they had to make multiple passes. Each pass through the problem solving process improved.

- IP – In Progress and does not meet our expectations. The student struggled with the concepts, either with the platform, syntax and/or data structure (if applicable). They were unable to complete the problem although they may have demonstrated portions of the solution that were correct. Overall their syntax should have been close. When asked to evaluate their design, they were unable to do so in a logical manner. Each pass did not necessarily improve.
- U – Unsatisfactory and does not meet our expectations. The student showed major issues with using the platform, editors, language and/or data structure (if applicable). Typically such students struggle with syntax issues and are performing operations that would not make sense for the problem at hand. Each pass through the problem does not improve and we end the proficiency demonstration when it is clear to us that the solution is not achievable.

Table 1. Scoring rubric for core CS competencies.

Competency	**Exceeds expectations**	**Proficient**	**In progress** *(non-passing)*	**Unsatisfactory**
Design Process	Design is well thought out with minimal revision necessary	A solid design was achieved; each pass improved	The design was close but flawed; each pass did not improve	The design was far from satisfactory; student was unable to design
Use of Recursion	Clear and concise	Correct use of recursion but could be simplified	Attempted recursion but with major flaws	Unable to solve a problem using recursion
Correct Syntax	Perfect syntax	Good syntax. Was able to correct errors with minimal assistance	Major syntactic issues. Most could be resolved but not all without assistance	Major syntactic issues which were not recoverable
Application of Data Structure Algorithms	Perfect use and/or implementation of data structure algorithms	Fluent with data structures but may have required multiple passes	Not fluent with the data structures; multiple passes did not improve	Major issues with the data structure algorithms. Far from functional

The process of evaluating student competencies has students independently solving randomly assigned problems while being observed by faculty and staff. Students' progress is closely monitored and assistants take notes during the process to evaluate overall performance in problem solving and syntax. Students have no access to course materials or the internet.

Only students that pass the PT may progress to the next course. Applicants transferring in at the upper division level must also pass in order to be admitted into the upper division program. Students that fail to pass are advised to take (or re-take) courses to gain the necessary competencies.

Preparing Students for Proficiency Testing

Students are prepared for the proficiency examinations by attending weekly labs where they have hands-on experience with the course concepts in a small group setting with approximately eight students per technical assistant. By midterm time, students experience the PT as a trial run, acquainting students with the process. Students that do not demonstrate the necessary programming competencies can re-demonstrate their competencies after one week of practice. Such a trial run gives students the confidence to program under observation, which may be a new process for them. Based on the findings at these trials, we can advise students on how best to be successful.

Results of Proficiency Tests

For each final proficiency examination processed, we tabulate the number of students that scored in each of the four categories, and for each of those how many missed the most important core competencies for the particular course being taken. These numbers are then averaged and reported to the department. This process is performed only for students that pass the course; our results do not include data from students that drop or fail the class for other reasons.

Our current results include the performance of over 2,700 students. Our findings show that the success of our students increased significantly from the first tests. At the start of our proficiency testing only 22% of students were judged to exceed expectations. Once the tests were fully established, we have seen this grow to 40-45% even for the most difficult courses. This is in stark contrast to the transfer students in summer 2015, of which only 5% demonstrated the same level of proficiency.

Our early test data showed that 2.48% of the students received an unsatisfactory grade even though they were passing all of the rest of the material, and 5-6% of students who were categorized as in-progress. In later years student performance significantly improved, so that in 2015 there are only 0.25% in unsatisfactory and 1% in in-progress categories, as shown in Table 2. We interpret this drop as significant improvement in student performance. Anecdotally, professors teaching upper-division courses have also noticed improved quality of student work in their courses. We expect these improvements to continue in 2016.

Table 2. Summary of the average student performance on CS proficiency tests since adopting the procedure.

Introductory Results	Exceed	Proficient	In Progress *Non-passing*	Unsatisfactory (*)
Early Averages:	22%	70%	5%	2.48%
2013 Averages:	45%	46%	6%	2.39%
2014 Averages:	39%	58%	3%	0.59%
2015 Averages:	40%	58%	1%	0.25%

(*) Received an Unsatisfactory Score on the Proficiency Demonstration but received passing scores on all other work. Failing grade in the class was due solely to the proficiency demonstration score.

Somewhat alarmingly, 48% of transfer students failed the proficiency examination during the summer 2015 tests. These students theoretically had all of the required prerequisites to immediately start courses at the upper division level, but could not demonstrate programming proficiency at our required level. Five percent of those who failed were advised to retake the freshman level CS162 course, setting them back one to one and a half years. We observed that they were deficient in the ability to use pointers and dynamic memory. Another 19% of students who failed were recommended to retake CS163 Data Structures because they did not fully understand lists and trees. In previous years these students would have been admitted automatically into the program and would suffer academically due to lack of preparation. Further comparison between native PSU and transfer students is given in Table 3**.**

Table 3. Deficiencies observed among CS transfer students relative to native students.

Problems with:	Basic programming	Recursion	Function calls	Data structure algorithms
Native (our U.)	0%	4%	2%	6%
Transfer	38%	29%	19%	29%

Additional Benefits

Proficiency-based examinations also have a positive impact on the overall curriculum. By evaluating the results of the students performing the demonstration, we have been able to fine tune our curriculum. For example, since 8% of our students were not properly using function returned values, we have modified our lab materials with weekly exercises requiring the use of returned values.

Overall, we believe that these results demonstrate the effectiveness of our approach to testing programming skills, and all of the expected benefits can be verified by data.

Competence-based Testing in EE

In our electrical engineering program, we have designed an introductory sequence of three quarter-long courses[11]. The second (ECE 102) and the third (ECE 103) deal with MATLAB and C programming, respectively. In addition, ECE 102 addresses engineering problem solving and utilizes MATLAB to drive a data-acquisition device as part of a major course project[12]. Two out of six course outcomes in ECE 102 deal with MATLAB programming. Course outcomes are assessed in homeworks, exams, labs and projects. In the rest of the curriculum, students are primarily using their existing programming skills, for example using C to program DSP chips. It is, therefore, critical that students get a solid foundation and practice in basic programming skills. Many students find programming very hard and end up with a piecemeal understanding of it[13,14]. They also “optimize” their efforts by devoting less time and effort to certain areas and compensating for it in others, with programming often being sidelined in such cases.

Our exploration of competency-based testing (CT) was motivated by these objectives:

- Ensuring that students develop a solid programming foundation
- Providing explicit and detailed guidance on what is expected
- Providing useful, timely, and frequent feedback to students
- Using CT results to improve the effectiveness of our teaching

Given that proficiency testing proved successful in attaining these and other goals, we believe that a somewhat simplified version of it, which we call competency-based testing, will accomplish the same in our classes. We have been experimenting with the content and format of our CT exams, and our initial findings are presented below.

Determining and Evaluating Student Competencies in EE

The programming competencies that we would like our students to exhibit are:

1. Variable usage
2. Vector manipulation
3. I/O functions
4. Branching
5. Loops
6. Function definition & calling

The first three are simple enough that they can be learned during the first two weeks of the course. Students must be comfortable using them before moving on to other topics. There are still some difficult concepts that students need to master even for this introductory material, e.g., assignment vs. equality and indexing within vectors. While we cannot lose sight of the need for students to understand these concepts, the ultimate goal is repeated practice and feedback to build and reinforce programming skills. At this stage, it may be difficult to distinguish various levels of performance, i.e., proficiency. The second half of the competency list presents a much higher level of conceptual difficulties, which is compounded by the need to integrate them into one programming and problem-solving whole. Testing for these competencies is particularly important before students take a follow-on course in C language programming, since these concepts are exploited more thoroughly there.

Testing for the first three competencies consists of simple programming tasks that take several lines of code to accomplish and are largely independent of each other. Students have to produce correct intermediate steps and final results. For the second half of the competencies list, we need to embed them in a larger problem which is still doable within 20-30 lines of code. Designing these problems is obviously more challenging, and we are learning how to scale our expectations to only the essential parts. In addition, we have to provide some variety to prevent students from memorizing a discrete set of problems that may come up on tests. We are in the process of developing a set of template problems to accommodate current and future CT needs. Finally, student competency is determined by direct observation of their programming performance during an in-class test.

Preparing Students for Competency Testing

Our approach to teaching programming is one based on active learning and giving students frequent and timely feedback. To accomplish this, we use a non-traditional e-book[15] that has many interactive problems which we assign as reading and monitor for student compliance. In-class activities include interaction systems[16] for collecting student answers and work. Most

recently, we added Cody Coursework exercises to supplement traditional MATLAB homework assignments. However, most of these activities deal with a small segment of the required skills or competencies, so in order to provide students with a chance to put it all together, we have now included a set of programming exercises that are done in a lab environment. Note that students also have to produce a much larger program which is part of their final project and is evaluated separately.

Results of Competency Tests

First ECE CT trial (Winter 2015)
In the winter quarter of 2015, the first implementation of the ECE CT was attempted on two sections of ECE 102 students. Given that this was our first attempt, we decided to make it voluntary and count only as extra credit with no make-up test offered. In total, 38 students volunteered to take the CT. While we cannot guarantee that this sample was representative, we observed that there was a cross-section of students in terms of their programming abilities making the analysis somewhat generalizable.

A set of basic test problems was developed by the instructors that could be solved in around twenty lines of code. The competency test was presented at the end of the academic quarter. Multiple testing sessions were offered, with each session being attended by both instructors and a teaching assistant (TA). Students were expected to bring their own laptop computer with MATLAB installed. Students were assigned a randomly selected test problem and given up to 25 minutes to solve it and demonstrate their work to the instructor. Access to the MATLAB help system was allowed just once. After completing the test, or when the time limit was reached, the instructors evaluated the student's results.

Second ECE CT trial (Spring 2015)
Another section of ECE 102 was taught in spring quarter, and this time all students were required to take the CT. Students were given two chances to pass the exam. To better prepare them, optional lab sessions were offered in which students practiced programming under TA and instructor supervision. Typically, only one quarter to one third of the class participated in the lab, which was deemed too low. The CT process was very similar to the previous one except the testing time was extended to one hour.

ECE CT Results for 2015
For the first CT trial, the combined passing percentage was a disappointing 55%. When a weighted composite grade based only on each student's MATLAB-specific class work was constructed, around 69% of the participants' CT results matched what would be predicted from their composite results. This means that there were significant number of students who passed CT but were not passing other MATLAB related coursework and vice-versa. We will need to collect more data and in a more systematic fashion before we can draw any conclusions from this observation. For now, it seems to indicate that CT can be useful in triangulating student success so that we do not rely on traditional assessments alone. The first-time results, however, were unsatisfactory and indicated that changes needed to be made in the course structure to improve student outcomes.

From this first attempt, a few observations were made:

- Several students were unable to complete the test within the initial 25-minute limit, so more time would need to be allocated for future trials.
- It is vital to explain clearly the instructor's expectations and the P/NP process at the start of the course.
- Direct observation of students working through the problems gave us indications where they struggled the most and what needed to be clarified or emphasized.
- Students were still not fully independent when programming and were particularly ill prepared for debugging.

For the second CT trial, 43 students took the CT, of which 30% failed initially and needed to take the make-up test. After their second try only 2% of students failed the CT. Recall that the first CT trial was treated as an optional extra credit assignment. In contrast, the second CT trial was required, so students could fail the class if they did not pass the CT. This makes the comparison between the final passing rates given in Table 4 difficult. However, we believe that improvements were at least in part due to increasing the test time and offering the optional labs. Based on this observation, our future classes will require programming labs and CT.

Table 4. ECE 102 Competency test results for 2015.

Quarter	Total Students	Passed (initial)	Passed (final)
Winter 2015	38	-	55%
Spring 2015	43	70%	98%

From the second round of CT trials, these lessons were learned:

- Having more assistants would make the process smoother and less time consuming.
- Programming labs with "live" help can help prepare students better, but we need to make the labs required.
- Two competency tests are needed so that students can get used to the format and ensure they know basic concepts such as variables and arrays before attempting more advanced topics like branching and loops.

Third ECE CT trial (Winter 2016)

After analyzing the results from the 2015 CT trials, the 2016 schedule was revised to offer two CT exams, one (CT-1) at the end of the third week and the other (CT-2) in the sixth week, with a make-up test offered after each CT. The first test covered variables, math operators, vector manipulation, basic plots, and calling functions. The second test assesses knowledge of comparison and logical operators, branching and loop statements, and writing of custom functions. From the experience with poor attendance in voluntary labs, a weekly lab class became mandatory. The instructors, teaching assistants (TAs), and undergraduate helpers attended each lab session to provide assistance during the programming exercises.

The two course sections were again taught by the same instructors. A total of 73 students took the CT-1 exam. Seven testing blocks of 45 minutes each were offered. This time, students were

tested on department Linux computers running MATLAB. Both course instructors and a TA performed assessment. Helpers assisted with the check-in and check-out duties. When the student was done, the instructor or TA looked over the code and output to decide if the student had passed the test. The TA was only allowed to give a passing score. If the TA believed the student failed, then an instructor reviewed the student's work to make the final decision.

At the time of this writing, only CT-1 data are available. Approximately 85% of the class passed the CT on the first try. After the make-up test results were factored in, 95% of the students were successful. Hence, the overall CT passing rate was much improved compared to the trial in 2015, though it needs to be mentioned that simpler material was tested.

Lessons Learned from the CT and Future Plans

After initial experimentation and refinements, we are now starting to approach steady-state in our efforts to establish competency testing as a viable assessment and teaching tool. Even though we had an example from CS to follow, we still have to make our own way through many of the obstacles. At this time, we only have our own observations to draw on, but our conclusion is that CT was a worthwhile investment that we will continue to refine. CT results do not mean much in isolation, so they need to be a part of a larger effort to develop student programming skills in an effective and efficient way. Because of this overall effort, we believe that students have reaped the benefits and will be much better prepared. Data to back up this claim will be collected over the coming years. One of our future projects will be a publication of a comprehensive manual that will cover various components, i.e., labs, Cody exercises and readings, and the CT itself.

Overall Conclusions

Based on our experiences and results of proficiency testing in computer science and competency testing in electrical engineering, we have reached the following conclusions:

- Proficiency testing has demonstrated improved student programming outcomes in our computer science program.
- Given the four-year record of implementation and success, proficiency testing is a good example to follow in electrical engineering in order to improve student programming skills.
- A simplified version of proficiency testing, labeled competency testing, may be sufficient for now in EE but may be expanded to full proficiency testing later on.
- Improvements in electrical engineering student learning have yet to be fully demonstrated, but initial results are encouraging.
- Proficiency and competency testing make expectations clear to students but have to be supplemented with other improvements in teaching.
- Proficiency and competency testing benefit students by making sure that they actually mastered the basics and can perform programming tasks before moving on to more complex concepts and courses.
- Other benefits, such as assessment of transfer students, may be obtained once the system is in place.
- Implementation does require additional resources in terms of trained TAs and helpers but has been shown to scale well to large numbers of students.

Proficiency testing has worked well, and we continue to develop it in collaboration with other universities and local high schools where proficiency testing is used for college credit. We hope that the descriptions and data presented here will encourage other programs to start experimenting with these testing techniques.

Bibliography

[1] ABET "Criteria for Accrediting Engineering Programs," http://www.abet.org/wp-content/uploads/2015/04/E001-14-15-EAC-Criteria.pdf , accessed Jan. 30, 2016

[2] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," ACM SIGCSE Bulletin, vol. 33, no. 4, pp. 125–180, 2001.

[3] R. S. Lemos, "Measuring Programming Language Proficiency," AEDS Journal, vol. 13, no. 4, pp. 261–273, Jun. 1980.

[4] M. J. Stehlik and P.L. Miller, "Implementing a mastery examination in computer science," 1985, downloaded from http://repository.cmu.edu/cgi/viewcontent.cgi?article=2555&context=compsci , accessed Jan 30, 2016.

[5] J. Carrasquel, "Competency Testing in Introductory Computer Science: The Mastery Examination at Carnegie-Mellon University," in Proceedings of the Sixteenth SIGCSE Technical Symposium on Computer Science Education, New York, NY, USA, 1985, p. 240–.

[6] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A Multi-national Study of Reading and Tracing Skills in Novice Programmers," in Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education, New York, NY, USA, 2004, pp. 119–150.

[7] M. E. Califf and M. Goodwin, "Testing Skills and Knowledge: Introducing a Laboratory Exam in CS1," in Proc. 33rd SIGCSE Technical Symposium on Computer Science Education, New York, NY, USA, 2002, pp. 217–221.

[8] J. Mead, S. Gray, J. Hamer, R. James, J. Sorva, C. S. Clair, and L. Thomas, "A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition," in Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education, New York, NY, USA, 2006, pp. 182–194.

[9] R. Klein-Collins, "Sharpening our focus on learning: The rise of competency-based approaches to degree completion," National Institute for Learning Outcomes Assessment, Occasional Paper, vol. 20, 2013.

[10] Education Department web site: http://www.ed.gov/oii-news/competency-based-learning-or-personalized-learning , accessed January 29, 2016.

[11] P. Wong, M. Holtzman, B. Pejcinovic, and M. Chrzanowska-Jeske, "Redesign of Freshman Electrical Engineering Courses for Improved Motivation and Early Introduction of Design," ASEE Annual Conference and Exhibition, Vancouver, Canada, 2011, pp. 22.1224.1 – 22.1224.13.

[12] P. Wong and B. Pejcinovic, "Teaching MATLAB and C Programming in First Year Electrical Engineering Courses Using a Data Acquisition Device," ASEE Annual Conference and Exhibition, Seattle, WA, 2015, pp. 26.1480.1 – 26.1480.11.

[13] T. Jenkins, “On the difficulty of learning to program,” in Proc. of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, 2002, vol. 4, pp. 53–58.

[14] M. J. Scott and G. Ghinea, “Educating programmers: A reflection on barriers to deliberate practice,” in Proc. 2nd HEA Conf. on Learning and Teaching in STEM Disciplines, 2013, p. 028P.

[15] zyBooks “Programming in MATLAB”, https://zybooks.zyante.com/#/catalog , accessed Jan. 30, 2016.

[16] Learning Catalytics from Pearson, https://learningcatalytics.com/ , accessed Jan. 30, 2016.